# PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2000-268074
(43)Date of publication of application : 29.09.2000

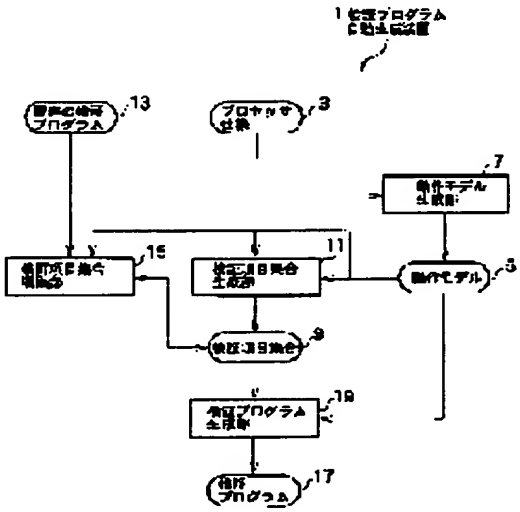
(51)Int.Cl. G06F 17/50
G01R 31/28
G06F 9/38

(21)Application number : 11-074118
(22)Date of filing : 18.03.1999

(71)Applicant : TOSHIBA CORP
(72)Inventor : KONO KAZUYOSHI
IMAI HIROSHI
MIZUNO ATSUSHI
KAMIYA HIRONORI

## (54) DEVICE AND METHOD FOR AUTOMATICALLY GENERATING VERIFICATION PROGRAM AND DEVICE AND METHOD FOR AUTOMATICALLY GENERATING PROPERTY

(57)Abstract:
PROBLEM TO BE SOLVED: To provide a device and a method for automatically generating a verification program and a property for verifying the design (operation) of a processor.
SOLUTION: An automatic verification program generator 1 is composed of an operation model generating part 7 for inputting a processor specification 3 and generating an operation model 5 of the processor, a verification item set generating part 11 for inputting the operation model 5 and the processor specification 3 and generating a verification item set, a verification item set editing part 15 for inputting an existent verification program 13 and the operation model 5 and editing a verification item set 9 and a verification program generating part 19 for inputting the verification item set 9 and the operation model 5 and generating a verification program 17 corresponding to individual verification items.




LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]
[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2000－268074
（P2000－268074A）
(43)公開日　平成12年９月29日(2000.9.29)

| (51)Int.Cl.' | 識別記号 | ＦＩ | | テーマコード'(参考) |
|---|---|---|---|---|
| Ｇ０６Ｆ　17/50 | | Ｇ０６Ｆ　15/60 | ６７０Ｋ | ２Ｇ０３２ |
| Ｇ０１Ｒ　31/28 | | 9/38 | ３５０Ａ | ５Ｂ０１３ |
| Ｇ０６Ｆ　9/38 | ３５０ | Ｇ０１Ｒ　31/28 | Ｆ | ５Ｂ０４６ |
| | | | | ９Ａ００１ |

審査請求　未請求　請求項の数４　ＯＬ　（全 13 頁）

(21)出願番号　特願平11－74118

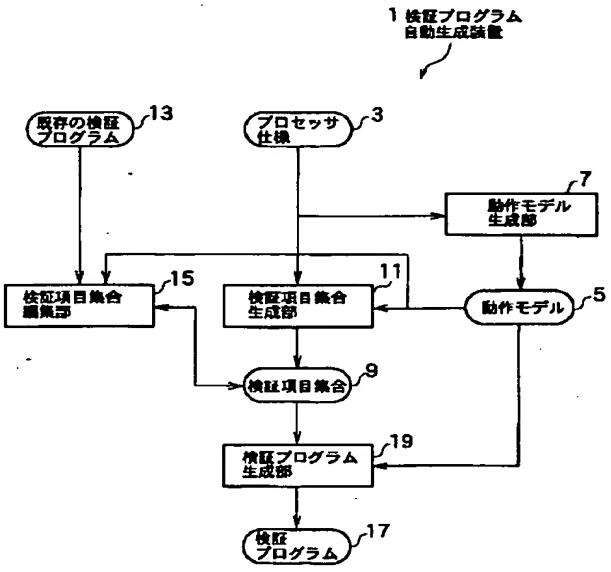
(22)出願日　平成11年３月18日(1999.3.18)

(71)出願人　000003078
株式会社東芝
神奈川県川崎市幸区堀川町72番地
(72)発明者　河野　和義
神奈川県川崎市幸区堀川町580番１号　株式会社東芝半導体システム技術センター内
(72)発明者　今井　浩史
神奈川県川崎市幸区堀川町580番１号　株式会社東芝半導体システム技術センター内
(74)代理人　100083806
弁理士　三好　秀和　（外７名）

最終頁に続く

(54)【発明の名称】　検証プログラム自動生成装置および方法並びにプロパティ自動生成装置および方法

(57)【要約】
【課題】　プロセッサの設計検証（動作検証）を行なうための検証プログラムおよびプロパティを自動生成する装置および方法を提供する。
【解決手段】　プロセッサ仕様３を入力し、プロセッサの動作モデル５を生成する動作モデル生成部７と、動作モデル５およびプロセッサ仕様３を入力し、検証項目集合を生成する検証項目集合生成部１１と、既存の検証プログラム１３および動作モデル５を入力し、検証項目集合９を編集する検証項目集合編集部１５と、検証項目集合９および動作モデル５を入力し、各検証項目に対応する検証プログラム１７を生成する検証プログラム生成部１９とから構成される検証プログラム自動生成装置１である。



【特許請求の範囲】
【請求項１】　プロセッサ仕様を入力し、プロセッサの動作モデルを生成する動作モデル生成部と、
前記動作モデルおよび前記プロセッサ仕様を入力し、検証項目集合を生成する検証項目集合生成部と、
既存の検証プログラムおよび前記動作モデルを入力し、前記検証項目集合を編集する検証項目集合編集部と、
前記検証項目および前記動作モデルを入力し、各検証項目に対応する検証プログラムを生成する検証プログラム生成部とを少なくとも有することを特徴とする検証プログラム自動生成装置。
【請求項２】　プロセッサ仕様を入力し、プロセッサの動作モデルを生成する動作モデル生成ステップと、
前記動作モデルおよび前記プロセッサ仕様を入力し、検証項目集合を生成する検証項目集合生成ステップと、
既存の検証プログラムおよび前記動作モデルを入力し、前記検証項目集合を編集する検証項目集合編集ステップと、
前記検証項目および前記動作モデルを入力し、各検証項目に対応する検証プログラムを生成する検証プログラム生成ステップとを少なくとも有することを特徴とする検証プログラム自動生成方法。
【請求項３】　プロセッサ仕様を入力し、プロセッサの動作モデルを生成する動作モデル生成部と、
その動作モデルおよびプロセッサの回路記述を入力し、前記プロセッサのプロパティを生成するプロパティ生成部とを少なくとも有することを特徴とするプロパティ自動生成装置。
【請求項４】　プロセッサ仕様を入力し、プロセッサの動作モデルを生成する動作モデル生成ステップと、
その動作モデルおよびプロセッサの回路記述を入力し、前記プロセッサのプロパティを生成するプロパティ生成ステップとを少なくとも有することを特徴とするプロパティ自動生成方法。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、プロセッサの設計検証を行なうための検証プログラムを自動生成する装置および方法に関する。また、本発明は、プロセッサの動作検証を行なうためのプロパティを自動生成する装置および方法に関する。
【０００２】
【従来の技術】高効率な検証プログラムの生成はプロセッサの設計検証容易化のためには不可欠である。しかし、検証プログラムの検証項目作成は人手によるため、検証項目の漏れ、誤った検証項目の作成などが起こり得る。さらに、プロセッサの論理が複雑になると検証項目の個数も膨大となり、人手での検証項目の作成はより一層困難となる。また、すでに作成済の検証プログラムが存在すれば、その検証プログラムの検証項目に関しては新たに検証プログラムを作成する必要はない。ところが、既存の検証プログラムが目的の検証項目を満たすものであるか否かは実際にその検証プログラムをプロセッサに与えてシミュレーションしてみなければわからない。したがって、検証プログラムの個数が多い場合もしくは検証プログラムのサイズが大きい場合には膨大なシミュレーション時間を要する。
【０００３】また、検証プログラムはプロセッサの動作モデルに基づいて作成されるが、その動作モデルの作成は設計者がプロセッサの仕様を参考に行なっている。したがって、動作モデル作成にはかなりの時間が必要であり、また、プロセッサの仕様が変更されればその仕様を満たすように動作モデルを再び作成し直す必要があった。
【０００４】一方、製品サイクルが短くなり、設計から市場投入までの時間の短縮が求められている。プロセッサの設計では設計中の論理回路の動作検証に多くの時間が費やされており、この時間短縮のため動作検証にシミュレーションを用いない形式的検証方法が採用されてきている。
【０００５】形式的検証方法のうちモデル検証と呼ばれる方法は設計対象のプロセッサの動作をプロパティとして記述し、プロセッサがプロパティを満足するかどうかを形式的に検証する。このようにして、設計対象の論理回路が仕様通りの動作を行うかどうかを検証する。
【０００６】従来、プロパティは設計者が仕様と回路記述を見ながら作成していたが、プロセッサの大規模化に伴いその仕様が複雑化し、プロパティの作成に要する時間が長期化する傾向にある。また、人手によるため、正しいプロパティを作成することが難しいという問題がある。
【０００７】
【発明が解決しようとする課題】本発明は、このような課題を解決し、プロセッサの設計検証を行なうための検証プログラムを自動生成する装置および方法を提供する。また、本発明は、プロセッサの動作検証を行なうためのプロパティを自動生成する装置および方法を提供する。
【０００８】
【課題を解決するための手段】上記課題を解決するため、本発明の第１の特徴は、プロセッサ仕様を入力し、プロセッサの動作モデルを生成する動作モデル生成部と、前記動作モデルおよび前記プロセッサ仕様を入力し、検証項目集合を生成する検証項目集合生成部と、既存の検証プログラムおよび前記動作モデルを入力し、前記検証項目集合を編集する検証項目集合編集部と、前記検証項目および前記動作モデルを入力し、各検証項目に対応する検証プログラムを生成する検証プログラム生成部とを少なくとも有する検証プログラム自動生成装置であることである。

【０００９】本発明の第２の特徴は、プロセッサ仕様を入力し、プロセッサの動作モデルを生成する動作モデル生成ステップと、前記動作モデルおよび前記プロセッサ仕様を入力し、検証項目集合を生成する検証項目集合生成ステップと、既存の検証プログラムおよび前記動作モデルを入力し、前記検証項目集合を編集する検証項目集合編集ステップと、前記検証項目および前記動作モデルを入力し、各検証項目に対応する検証プログラムを生成する検証プログラム生成ステップとを少なくとも有する検証プログラム自動生成方法であることである。

【００１０】本発明の第３の特徴は、プロセッサ仕様を入力し、プロセッサの動作モデルを生成する動作モデル生成部と、その動作モデルおよびプロセッサの回路記述を入力し、前記プロセッサのプロパティを生成するプロパティ生成部とを少なくとも有するプロパティ自動生成装置であることである。

【００１１】本発明の第４の特徴は、プロセッサ仕様を入力し、プロセッサの動作モデルを生成する動作モデル生成ステップと、その動作モデルおよびプロセッサの回路記述を入力し、前記プロセッサのプロパティを生成するプロパティ生成ステップとを少なくとも有するプロパティ自動生成方法であることである。

【００１２】

【発明の実施の形態】（第１の実施の形態）図１は本発明の第１の実施の形態に係る検証プログラム自動生成装置の構成を示すブロック図である。本発明の第１の実施の形態は、パイプライン化したプロセッサを設計する場合に本発明を適用した例である。図１に示すように、本発明の第１の実施の形態に係る検証プログラム自動生成装置１は、プロセッサの仕様３を入力し、プロセッサの動作モデル５を生成する動作モデル生成部７と、プロセッサの仕様３および動作モデル５を入力し、検証項目集合９を生成する検証項目集合生成部１１と、既存の検証プログラム１３および動作モデル５を入力し、検証項目集合９を編集する検証項目集合編集部１５と、動作モデル５および検証項目集合９を入力し、各検証項目に対応する検証プログラム１７を生成する検証プログラム生成部１９とを備えて構成される。

【００１３】プロセッサの仕様３には設計対象のプロセッサの命令セット・アーキテクチャ（ＩＳＡ：instruction set architecture）仕様、各命令のパイプライン動作仕様、パイプラインのストール（stall）規則が含まれる。図２に命令セット・アーキテクチャ仕様の例を示す。図２に示した例では、命令セット・アーキテクチャ仕様として６個の命令およびその動作が与えられている。「ｎｏｐ」命令は何も行なわない命令、「ａｄｄ」命令は整数の加算を行なう命令、「ｓｕｂ」命令は整数の減算を行なう命令、「ｆａｄｄ」命令は浮動小数点数の加算を行なう命令、「ｆｓｕｂ」命令は浮動少数点数の減算を行なう命令、「ｌｄ」命令はメモリからデータをロードする命令である。

【００１４】図３にパイプライン動作仕様の例を示す。パイプライン制御の動作を考えた場合に全く同様に動作する命令はグループ化することができるようにしておく。図３に示した例では、「ａｄｄ」命令および「ｓｕｂ」命令を「ｉｎｔ」命令に、「ｆａｄｄ」命令および「ｆｓｕｂ」命令を「ｆｐｕ」命令にグループ化する。各パイプライン・ステージにおける動作（パイプライン動作）は次の通りである。「ＦＥＴ」ステージは命令のフェッチ、デコードおよびレジスタ・フェッチを行なう。「ＡＬＵ」ステージは整数演算の実行を行なう。「ＦＰＵ」パイプライン・ステージは浮動小数点演算の実行を行なう。「ＭＥＭ」ステージはメモリ・アクセスを行なう。「ＷＢ」ステージはレジスタへの書き込みを行なう。

【００１５】パイプラインのストール規則は構造ハザード・ストール規則とデータ・ハザード・ストール規則からなる。構造ハザード・ストール規則はパイプライン動作中に構造ハザードが起こった場合のストール規則を記述する。データ・ハザード・ストール規則はパイプライン動作中にデータ・ハザードが起こった場合のストール規則を記述する。なお、ストール規則については後で詳しく説明する。

【００１６】（動作モデル生成部７）動作モデル生成部７はプロセッサ仕様３からプロセッサの動作モデル５を生成する。生成された動作モデル５は検証項目集合生成部１１、検証項目集合編集部１５および検証プログラム生成部１９で使用される。図４に動作モデル生成部７の動作を示すフローチャートを示す。動作モデル生成部７の処理は、（１）パイプライン構造グラフの作成（ステップ４１）、（２）構造ハザードによるストール条件の決定（ステップ４２）、（３）データ・ハザードによるストール条件の決定（ステップ４３）、（４）有限状態機械（finite state machine）の生成（ステップ４４）、に分かれる。

【００１７】（１）パイプライン構造グラフの作成

まず、動作モデル生成部７はパイプライン構造グラフを作成する。パイプライン構造グラフはパイプラインを構成する各ステージの接続関係を表す有向グラフＧである。Ｇはパイプライン動作仕様に記されているパイプライン・ステージを節点、パイプライン・ステージ間の接続を有向辺とし、同一のパイプライン・ステージを表す節点を一つにまとめることにより作成される。図５に図３に示したパイプライン動作仕様から得られるパイプライン構造グラフを示す。

【００１８】（２）構造ハザードによるストール条件の決定

次に、動作モデル生成部７は構造ハザードによるストール条件を決定する。ステップ１で求めたパイプライン構造グラフの節点のうち複数の有向辺の終点となる節点が

構造ハザードの原因となる。たとえば図５に示したパイプライン構造グラフでは、３個の有向辺の終点となる節点「ＷＢ」が構造ハザードの原因となる。構造ハザードによるストール条件は、まず構造ハザード状態集合を求め、この構造ハザード状態集合と構造ハザードストール規則を用いて構造ハザードによるストール条件を決定する。

【００１９】構造ハザード状態集合は、パイプライン構造グラフから２個以上の有向辺の終点となる節点の集合Ｓを求め、集合Ｓのすべての要素ｓについて次の処理を行なうことで求められる。まず、要素ｓを終点に持つ有向辺の集合Ｅを求める。そして、集合Ｅの要素ｅのすべての対（$e_i$，$e_j$）（但し、（$e_i$，$e_j$）と（$e_j$，$e_i$）は同一視する）について、$e_i$の始点を$s_i$、$e_j$の始点を$s_j$とし、有効グラフＧの節点ｎに対応するパイプラインステージに来る可能性のある命令の集合をＩｎｓｔ（ｎ）とする。Ｉｎｓｔ（$s_i$）のすべての要素$i_k$およびＩｎｓｔ（$s_j$）のすべての要素$j_l$について（（$s_i$，$i_k$），（$s_j$，$j_l$））を構造ハザード状態集合の要素に加える。ステージが異なっていても、同一のリソースを使用する場合、構造ハザードが起こる。このような場合も、ほぼ同様の方法を用いることにより、構造ハザード状態を求めることが可能である。図６に図３に示したパイプライン動作仕様に対する構造ハザード状態集合を示す。

【００２０】構造ハザード状態集合が得られたら、構造ハザード状態集合のすべての要素についてハザードが起きた時の動作を決定し、すべての構造ハザード状態集合の要素に対応するパイプラインの状態に対して構造ハザードが起こった時のストール条件を決定する。この時に構造ハザード・ストール規則を用いる。図７に構造ハザード・ストール規則の例を示す。この例では、命令間に優先順位を決めておき優先順位の最も高いもの以外の命令がストールするという規則を表している。例えば、「（（ＡＬＵ，ｉｎｔ），（ＦＰＵ，ｆｐｕ））－（ＡＬＵ，Ｉｎｔ）」は、ＡＬＵステージにｉｎｔ命令が存在し、ＦＰＵステージにｆｐｕ命令が存在する場合、ＡＬＵステージに存在するｉｎｔ命令がストールすることを表している。図８に図７の規則から得られた構造ハザードによるストール条件を示す。たとえば図８の第２列は、ＡＬＵステージに存在するｉｎｔ命令が構造ハザードによりストールする条件は“ＭＥＭステージにｌｄ命令が存在するかＦＰＵステージにｆｐｕ命令が存在する時である”ことを表している。また第３列は、ＭＥＭステージに存在するｌｄ命令は構造ハザードによりストールすることはないことを表している。

【００２１】図９に構造ハザード・ストール規則の他の例を示す。この例では、構造ハザードを引き起こす命令の命令発行順序を比較し、発行順が最も早い命令以外の命令がストールするという規則を表している。$PS_i$、$PS_j$をパイプライン・ステージとしたとき、Ｏｒｄｅｒ（$PS_i$）＞Ｏｒｄｅｒ（$PS_j$）が真とは、$PS_i$に存在する命令が$PS_j$に存在する命令より後に発行されたことを表わすとする。図１０に図９の規則から得られた構造ハザードによるストール条件を示す。

【００２２】なお、本発明の第１の実施の形態において、決定した構造ハザードによるストール条件の編集（変更）機能を有していることが好ましい。構造ハザード・ストール規則では表現できないものや構造ハザード・ストール規則の例外等を扱うことができるようにするためである。

【００２３】（３）データ・ハザードによるストール条件の決定

次に、動作モデル生成部７はデータ・ハザードによるストール条件を決定する。まず、命令セット・アーキテクチャ仕様からデータ・ハザードが起こる命令対の集合（以下、データ・ハザード状態集合と呼ぶ）を求める。図３のパイプライン動作仕様ではＦＥＴステージでレジスタ・フェッチが行なわれるため、データ・ハザードのチェックはＦＥＴステージにおいて行なう。データ・ハザード状態集合の要素を（Ｉｎｓｔ（ＦＥＴ），（ＰＳ，Ｉｎｓｔ（ＰＳ））とする。データ・ハザードは先行命令のデスティーネーション・レジスタと後続命令のソース・レジスタが一致する場合に起こる。したがって、図２の命令セット・アーキテクチャ仕様および図３のパイプライン動作仕様の場合、データ・ハザード状態集合は図１１に示すものとなる。

【００２４】データ・ハザード状態集合が得られたら、データ・ハザード状態集合のすべての要素についてハザードが起きた時の動作を決定し、各命令対に対してデータ・ハザードによるストール条件を決める。この時にデータ・ハザード・ストール規則を用いる。図１２にデータ・ハザード・ストール規則の例を示す。この例では、○印がついている命令対でデータ・ハザードが起こった場合、ＦＥＴステージに存在する命令がストールすることを表している。また、図１３にデータ・ハザード・ストール規則の他の例を示す。この例では、パイプライン・ステージの前半でレジスタへの書き込みを行ない、パイプライン・ステージの後半でレジスタの参照を行なうという実装にすることで、ＷＢステージで書き込まれた値を同一サイクルのＦＥＴステージで参照可能となり、ＷＢステージに存在する命令との間でデータ・ハザードによるストールが起こらなくなる場合を示している。さらに図１４にデータ・ハザード・ストール規則の他の例を示す。この例は、フォワーディングを行なうことにより「ｌｄ」命令以外の命令でデータ・ハザードによるストールが起こらなくなる場合である。

【００２５】図１５に図１２の規則から得られたデータ・ハザード・ストール条件を示す。図１５において、「Ｉｎｓｔ（ＰＳ）」はパイプライン・ステージＰＳに

現在存在する命令を返す関数である。「ＤＨＳｔａｌｌＲｕｌｅ」はＦＥＴステージに存在する命令、ＦＥＴ以外のステージおよびそのステージに存在する命令から、ＦＥＴステージに存在する命令がデータ・ハザードによりストールする可能性があるか否かを返す関数であり、図１２～図１４に示したデータ・ハザード・ストール規則で対応するものが○印の場合“１”を返し、×印の場合“０”を返す。「ｓｒｃ」は命令のソース・レジスタの集合を返す関数であり、「ｄｅｓｔ」は命令のデスティネーション・レジスタを返す関数である。

【００２６】なお、本発明の第１の実施の形態において、決定したデータ・ハザードによるストール条件の編集（変更）機能を有していることが好ましい。データ・ハザード・ストール規則では表現できないものやデータ・ハザード・ストール規則の例外等を扱うことができるようにするためである。

【００２７】（４）有限状態機械（finite state machine）の生成

次に、動作モデル生成部７は各パイプライン・ステージの命令の組み合わせを状態とみなした有限状態機械を生成する。以下では説明簡略化のため、構造ハザードによるストール条件については図７に示した構造ハザード・ストール規則を用いて決定するものとする。一方、データ・ハザードによるストールについてはここでは考えないものとする。以降では、後段のパイプライン・ステージがストールしたらストールするというストール条件を「暗黙のストール条件」と呼ぶことにする。パイプライン・ステージＰＳのストール条件を「Ｓｔａｌｌ（ＰＳ）」、パイプライン・ステージＰＳの暗黙のストール条件を「ＩｍｐｌｉｃｉｔＳｔａｌｌ（ＰＳ）」、パイプライン・ステージＰＳの構造ハザード・ストール条件を「ＳＨＳｔａｌｌ（ＰＳ）」、パイプライン・ステージＰＳのデータ・ハザード・ストール条件を「ＤＨＳｔａｌｌ（ＰＳ）」で表すことにする。

【００２８】図１６に「Ｓｔａｌｌ（ＰＳ）」を表す式を示す。また、図１７に「ＩｍｐｌｉｃｉｔＳｔａｌｌ（ＰＳ）」を表す式を示す。但し、図１７において、ＰＳＳｅｔは全パイプライン・ステージからなる集合である。また、「Ｎｅｘｔ（ＰＳ，Ｉ））」はパイプライン・ステージＰＳに存在する命令がＩである場合の次段のパイプライン・ステージを返す関数である。図１８は図３のパイプライン動作仕様に対する関数Ｎｅｘｔを表で示したものである。また、図１９は論理式で示したものである。これらはパイプライン動作仕様から自動的に生成することができる。

【００２９】パイプライン・ステージＰＳに存在する命令がＩである時の構造ハザード・ストール条件を「ＳＨＳｔａｌｌ（ＰＳ，Ｉ）」とすれば、「ＳＨＳｔａｌｌ（ＰＳ）」は図２０で示される式で表される。またパイプライン・ステージＰＳに存在する命令がＩである時のデータ・ハザード・ストール条件を「ＤＨＳｔａｌｌ（ＰＳ，Ｉ）」とすれば、「ＤＨＳｔａｌｌ（ＰＳ）」は図２１で示される式で表される。但し、図２０、図２１において、ＩｎｓｔＳｅｔは全命令からなる集合である。図７に示した構造ハザードストール規則から導かれる「ＳＨＳｔａｌｌ（ＰＳ，Ｉ）」を図２２に示す。この「ＳＨＳｔａｌｌ（ＰＳ，Ｉ）」は図４のステップ２で求めた構造ハザード・ストール条件である。図２０に示した「ＳＨＳｔａｌｌ（ＰＳ）」の式を用いれば、「ＳＨＳｔａｌｌ（ＰＳ）」は図２３に示すものとなる。一方、データ・ハザードによるストールについては考慮しないので、「ＤＨＳｔａｌｌ（ＰＳ）」は図２４に示すものとなる。

【００３０】パイプライン構造グラフＧの節点間に以下に示す関係を定義することにより節点集合は半順序集合となる。

【００３１】定義１：パイプライン構造グラフＧを有向グラフとし、ｅをＧの有向辺とする。ｍがｅの始点、ｎがｅの終点のときｍ＞ｎである。

【００３２】「ＩｍｐｌｉｃｉｔＳｔａｌｌ（ＰＳ）」および「Ｓｔａｌｌ（ＰＳ）」を定義１の順序に従って作成すれば、すなわち「ＩｍｐｌｉｃｉｔＳｔａｌｌ（ＰＳ）」および「Ｓｔａｌｌ（ＰＳ）」作成時にＰＳより小さいパイプライン・ステージＰＳｊの「ＩｍｐｌｉｃｉｔＳｔａｌｌ（ＰＳｊ）」および「Ｓｔａｌｌ（ＰＳｊ）」が作成済みであるようにすれば、「ＩｍｐｌｉｃｉｔＳｔａｌｌ（ＰＳ）」および「Ｓｔａｌｌ（ＰＳ）」を求めることができる。たとえば図３に示したパイプライン動作仕様の場合は、ＷＢ→ＭＥＭ→ＦＰＵ→ＡＬＵ→ＦＥＴの手順で求めることができる。その結果を、図２５、図２６、図２７に示す。

【００３３】パイプライン・ステージＰＳの次状態関数「ＮｅｘｔＩｎｓｔ（ＰＳ）」を求めるにはパイプライン・ステージＰＳに対応する節点を以下の３つのケースに分けて考える。

【００３４】ケース１：節点を終点としてもつ有向辺が存在しない場合

ケース２：節点を始点としてもつ有向辺が存在しない場合

ケース３：上記以外の場合

有向グラフＧが１個の節点のみからなる場合にはケース１かつケース２という状況が起こり得るが、この場合グラフはパイプラインを表しているとはいえない。したがって、パイプラインを考える場合は必ず３つのケースは互いに独立となる。ケース１の場合「ＮｅｘｔＩｎｓｔ（ＰＳ）」は図２８に示す式、ケース２の場合「ＮｅｘｔＩｎｓｔ（ＰＳ）」は図２９に示す式、ケース３の場合「ＮｅｘｔＩｎｓｔ（ＰＳ）」は図３０に示す式となる。但し、図１８において、「Ｉｎｓｔ（ＩＮ）」は次サイクルに入力される命令を表わすものとし、図２９、

図３０において、「Ｂｕｂｂｌｅ」はステージに命令が存在しないことを表わすものとする。さらに、図２９、図３０において、「ＰＳＳｅｔ」は全パイプライン・ステージからなる集合であるとする。また、図３１に図２の命令セット・アーキテクチャ仕様および図３にパイプライン動作仕様に対して実際に求めた次状態関数を示す。

【００３５】このようにして動作モデル生成部７は命令セット・アーキテクチャ仕様、パイプライン動作仕様、構造ハザード・ストール規則およびデータ・ハザード・ストール規則を含むプロセッサ仕様３から動作モデル５（本実施の形態では有限状態機械である）を自動的に生成する。したがって、プロセッサの仕様３に変更があっても、プロセッサ仕様３を修正するだけで、変更されたプロセッサ仕様３に対する動作モデル５を自動的に生成することが可能となる。また、パイプラインのストール動作を変更したい場合でも、ストール規則を修正することにより、変更されたストール動作に対する動作モデルを自動的に生成できる。

【００３６】（検証項目集合生成部１１）次に、検証項目集合生成部１１の動作について説明する。検証項目は各パイプライン・ステージの状態の組み合わせで表現する。具体的にはたとえば図３２に示した通りである。検証項目集合生成部１１は構造ハザードが起こる各パイプライン・ステージの状態、データ・ハザードが起こる各パイプライン・ステージの状態を列挙する。なお、本実施の形態では、動作モデル生成部７がストール条件を生成する際にストールが起こるパイプラインの状態の組み合わせを求めている。したがって、検証項目集合生成部１１は動作モデル生成部７に含まれることになる。

【００３７】（検証項目集合編集部１５）次に、検証項目集合編集部１５の動作について説明する。検証項目集合編集部１５は検証項目生成部１１で作成された検証項目集合９を編集する。検証項目集合編集部１５の主な目的は既存の検証プログラム１３が存在すれば、既存の検証プログラム１３で検証可能な検証項目を検証項目集合９から取り除くことである。検証項目集合９を編集することにより検証プログラム生成部１９において不要な検証項目についての検証プログラムの生成を抑制することができる。

【００３８】検証項目集合編集部１５は既存の検証プログラム１３および動作モデル５に基づいて検証項目集合９を編集する。検証項目集合編集部１５は既存の検証プログラム１３を動作モデル５に与え、簡易シミュレーションを行なう。動作モデル５ではパイプラインの制御動作のみをモデル化しているため、オペランドで与えられるデータがどのように加工されるかはシミュレートしない。したがって、通常のシミュレーションと比べて高速に実行できる。各サイクル毎に各パイプライン・ステージの状態を記録し、その状態が検証項目集合９に含まれる場合には検証項目集合９からその状態を取り除く。

【００３９】（検証プログラム生成部１９）次に、検証プログラム生成部１９の動作について説明する。検証プログラム生成部１９は動作モデル５および検証項目集合９から検証項目集合９の各要素を検証するための命令列（検証プログラム）１７を自動生成する。検証項目は各パイプライン・ステージに存在する命令により表現されている。また動作モデル５は各パイプライン・ステージに存在する命令を状態とみなした有限状態機械として表現されている。これらを用いた命令列の自動生成は、ＢＤＤを用いることにより初期状態から到達可能な状態集合を計算することで、容易に実現可能である。

【００４０】本発明の第１の実施の形態によれば、プロセッサ仕様から自動的に動作モデルを生成できる。それにより動作モデル作成にかかる時間を短縮することができる。また、プロセッサの仕様に変更があった場合、命令セット・アーキテクチャ仕様もしくはパイプライン仕様を修正することにより、変更された仕様に対する動作モデルを自動的に生成することが可能となる。またパイプラインのストール動作を変更したい場合でも、ストール規則を修正することにより、変更されたストール動作に対する動作モデルを自動的に生成することが可能となる。

【００４１】また、本発明の第１の実施の形態によれば、従来人手により作成していた検証項目をプロセッサ仕様から自動的に生成することができる。このことにより検証項目の洩れを防ぐことができる。また人手作成に比べて、検証項目の作成に要する時間を減らすことができる。この効果は回路の複雑さが増せば増すほど顕著に表れる。

【００４２】さらに、本発明の第１の実施の形態によれば、既存の検証プログラムにより検証される検証項目を仕様から作成した検証項目集合から取り除くことができる。このことにより、不要な検証プログラムの生成を抑制することができる。ランダムなテストプログラム自動生成をアルゴリズミックなテストプログラム自動生成を組み合わせて行なう時にも有効である。

【００４３】（第２の実施の形態）次に、本発明の第２の実施の形態について説明する。図３３は、本発明の第２の実施の形態は、第１の実施の形態同様、パイプライン化したプロセッサを設計する場合に本発明を適用した例である。図３３に示すように、本発明の第２の実施の形態に係るプロパティ自動生成装置２１は、プロセッサ仕様２３を入力し、プロセッサ仕様２３を解析する仕様解析部２５と、その解析結果を入力し、動作モデルを生成する動作モデル生成部２７と、プロセッサの回路記述２９を入力し、プロセッサの回路記述２９を読み込む回路記述読み込み部３１と、その読み込み結果を入力し、プロセッサの回路記述２９を解析する記述解析部３３と、動作モデル生成部２７が生成した動作モデルと記述

解析部３３の解析結果を入力し、プロセッサのプロパティ３５を生成するプロパティ生成部３７とを備えて構成される。

【００４４】プロセッサの仕様２３には、図１に示した第１の実施の形態のプロセッサ仕様３と同様、設計対象のプロセッサの命令セット・アーキテクチャ（ＩＳＡ）仕様、各命令のパイプライン動作仕様、パイプラインのストール規則が含まれる。仕様解析部２５はプロセッサ仕様２３を解析し、動作モデル生成部２７が必要とする情報を提供する。仕様解析部２５が提供する情報に基づいて動作モデル生成部２７は動作モデルを生成する。本実施の形態では、仕様解析部２５が最初にプロセッサ仕様２３を解析しているが、第１の実施の形態のように動作モデル生成部２７が直接プロセッサ仕様２３を入力する構成としてもよい。

【００４５】プロセッサの回路記述２９はたとえばレジスタ転送レベル（ＲＴＬ）で記述される。回路記述読み込み部３１は回路記述２９を読み込み、記述解析部３３がプロパティ生成部３７が必要とする情報を回路記述２９から取り出す。プロパティ生成部３７は動作モデル生成部２７が生成した動作モデルと記述解析部３３が提供する回路記述についての情報からプロパティ３５を自動生成する。

【００４６】次に、図３３に示した動作モデル生成部２７の動作モデル生成とプロパティ生成部３７のプロパティ生成について説明する。図３４は、図３３の動作モデル生成部２７の動作モデル生成とプロパティ生成部３７のプロパティ生成の処理手順を示すフローチャートである。まず、動作モデル生成（ステップ３４２）は図１に示した本発明の第１の実施の形態の動作モデル生成部７の動作と同様であるのでここでは説明しない。また、ハザード状態生成（ステップ３４１）は構造ハザードが起こる各パイプライン・ステージの状態、データ・ハザードが起こる各パイプライン・ステージの状態を列挙する。本第２の実施の形態ではストール条件を生成する際、ストールが起こるパイプラインの状態の組み合わせを求めているため、動作モデル生成（ステップ３４２）でハザード状態を生成している。ハザード状態は各パイプライン・ステージの状態の組み合わせで表現される。具体的には図３２に示した通りである。なお、検証対象動作３９はたとえばパイプラインのハザードである。

【００４７】次に、ハザード解消状態生成（ステップ３４３）の処理について説明する。ハザード状態生成（ステップ３４１）で構造ハザード状態集合およびデータ・ハザード状態集合が求められている。また、動作モデル生成（ステップ３４２）で各パイプライン・ステージの命令の組み合わせを一つの状態とみなした有限状態機械が生成されている。したがって、ハザード状態と有限状態機械の次状態関数「Ｎｅｘｔ Ｉｎｓｔ」を用いてハザード解消状態を求めることができる。具体的には、構造ハザード状態集合ＳＨＳＳとデータ・ハザード状態集合ＤＨＳＳの各状態ｓに対して、まず各パイプライン・ステージＰＳｉに対してＮｅｘｔＩｎｓｔ（ＰＳｉ）を計算し、ハザード解消状態$H_s$に各ＮｅｘｔＩｎｓｔ（ＰＳｉ）を要素として加えることで求めることができる。図３５に、その例に対応したハザード解消状態を図３６に示す。

【００４８】次に、レジスタ名とシグナル名のマッチング（ステップ３４４）の処理について説明する。レジスタ名とシグナル名のマッチング（ステップ３４４）ではハザード状態生成（ステップ３４１）およびハザード解消状態生成（ステップ３４３）で生成された状態、すなわち各パイプライン・ステージの命令とプロセッサのＲＴＬ記述４１内のレジスタおよびシグナルとの対応付けを行う。例えば、パイプライン・ステージＦＥＴに命令ｉｎｔが存在している場合、ＲＴＬ記述４１中ではレジスタＦｅｔ＿ｒｅｇ［３：０］に値０００１が格納されている状態がパイプライン・ステージＦＥＴに命令ｉｎｔが存在する場合に対応するとき、「Ｉｎｓｔ（ＦＥＴ）＝＝ｉｎｔ」と「Ｆｅｔ＿ｒｅｇ［３：０］＝“０００１”」の対応付けを行う。

【００４９】次に、プロパティ生成（ステップ３４５）の処理について説明する。例えば、「ハザード状態ｓにあるとき、次のクロック・サイクルでハザード解消状態ｓ’になる」という型のプロパティを生成する場合を考える。図３５および図３６に示した例で説明する。各パイプライン・ステージの命令とＲＴＬ記述中のレジスタとの対応関係の例を図３７に示す。図３７には図３５および図３６に示した例で必要な部分のみを示した。プロパティ生成（ステップ３４５）は、ハザード状態生成（ステップ３４１）およびハザード解消状態生成（ステップ３４３）で生成された状態、すなわち各パイプライン・ステージの命令の組み合わせを、レジスタ名とシグナル名マッチング（ステップ３４４）で対応付けられた各パイプライン・ステージの命令とＲＴＬ記述４１中のレジスタの対応関係を用いて、ＲＴＬ記述４１中のレジスタに置き換えることによりプロパティ４３を生成する。図３５および図３６に示した例に対するプロパティ４３を図３８に示す。

【００５０】本発明の第２の実施の形態によれば、大規模プロセッサの動作検証の際に必要な検証対象動作を表すプロパティをプロセッサ仕様とプロセッサの回路記述から自動的に作成することができる。それにより短期間で動作検証のためのプロパティを用意することが可能となる。

【００５１】（第３の実施の形態）次に、本発明の第３の実施の形態について説明する。本発明の第３の実施の形態は、第２の実施の形態で示した動作モデル生成とプロパティ生成の他の例を示すものである。本発明の第３の実施の形態は、バス調停を設計する場合に本発明を適

用した例である。図３９に本第３の実施の形態に用いるプロセッサ仕様２３の例を示す。この例はバス調停のアルゴリズムがラウンド・ロビンであり、バスに接続されるコンポーネント（クライアントと呼ぶ）を示している。図４０に本第３の実施の形態に用いる検証対象動作３９の例を示す。ここでは、図３９および図４０に示した例に対してプロパティを自動生成する場合について説明する。図４１は、本発明の第３の実施の形態の動作を示すフローチャートである。

【００５２】図３９に示したラウンド・ロビン型のバス調停とは、バス使用要求を出したクライアントを待ち行列に置かれた順に一定時間ずつ処理し、クロック・サイクル毎に待ち行列の先頭にあったものを最後尾に置く調停方式である。まず、動作モデル生成（ステップ４１１）でラウンド・ロビン型バス調停を実現する上で必要なクライアント数、各クライアントの優先順位を生成する。その例を図４２に示す。この例では、クライアント数は８であり、クライアントの優先順位は適当に生成した。

【００５３】次に、レジスタ名とシグナル名のマッチング（ステップ４１２）では、動作モデル生成（ステップ４１１）で生成したモデルの各クライアントのバス使用要求、使用許可とＲＴＬ記述４１内のレジスタ名との対応関係を生成する。ＲＴＬ記述４１の例を図４３に、クライアント名とレジスタ名の対応関係の表を図４４に示す。

【００５４】最後に、プロパティ生成（ステップ４１３）では、「クライアントＣＬ０がバスをリクエストするとき、８クロック・サイクル以内に許可される」という型のプロパティを生成すると仮定すれば、レジスタ名、シグナル名マッチングの処理で得られた対応関係の表を用いてクライアント名をＲＴＬ記述中のレジスタ名に置き換え、プロパティ４３を生成する。生成されたプロパティの例を図４５に示す。

【００５５】本発明の第３の実施の形態においても第２の実施の形態と同様の効果を得ることができる。

【００５６】

【発明の効果】本発明によれば、プロセッサの動作検証を短時間で行なうことが可能となる。したがって、プロセッサの設計時間の短縮を図ることができる。

【図面の簡単な説明】

【図１】本発明の第１の実施の形態に係る検証プログラム自動生成装置の構成を示すブロック図である。

【図２】命令セット・アーキテクチャ仕様の例を示す図である。

【図３】パイプライン動作仕様の例を示す図である。

【図４】図１の動作モデル生成部の動作を示すフローチャートである。

【図５】図３のパイプライン動作仕様から得られるパイプライン構造グラフを示す図である。

【図６】図３のパイプライン動作仕様に対する構造ハザード状態集合を示す図である。

【図７】構造ハザード・ストール規則の例を示す図である。

【図８】図７の規則から得られた構造ハザードによるストール条件を示す図である。

【図９】構造ハザード・ストール規則の他の例を示す図である。

【図１０】図９の規則から得られた構造ハザードによるストール条件を示す図である。

【図１１】図２の命令セット・アーキテクチャ仕様および図３のパイプライン動作仕様に対するデータ・ハザード状態集合を示す図である。

【図１２】データ・ハザード・ストール規則の例を示す図である。

【図１３】データ・ハザード・ストール規則の他の例を示す図である。

【図１４】データ・ハザード・ストール規則の他の例を示す図である。

【図１５】図１２の規則から得られたデータ・ハザード・ストール条件を示す図である。

【図１６】パイプライン・ステージＰＳのストール条件「Ｓｔａｌｌ（ＰＳ）」を表す式を示す図である。

【図１７】パイプライン・ステージＰＳの暗黙のストール条件「ＩｍｐｌｉｃｉｔＳｔａｌｌ（ＰＳ）」を表す式を示す図である。

【図１８】図３のパイプライン動作仕様に対する関数Ｎｅｘｔを表で示した図である。

【図１９】図３のパイプライン動作仕様に対する関数Ｎｅｘｔを論理式で示した図である。

【図２０】パイプライン・ステージの構造ハザード・ストール条件を表す式を示す図である。

【図２１】パイプライン・ステージのデータ・ハザード・ストール条件を表す式を示す図である。

【図２２】各パイプライン・ステージの全命令に対する構造ハザード・ストール条件を表す表を示す図である。

【図２３】各パイプライン・ステージの構造ハザード・ストール条件を表す表を示す図である。

【図２４】各パイプライン・ステージに対応するデータ・ハザード・ストール条件を表す表を示す図である。

【図２５】各パイプライン・ステージの全命令に対する暗黙のストール条件を表す表を示す図である。

【図２６】各パイプライン・ステージに対する暗黙のストール条件を表す表を示す図である。

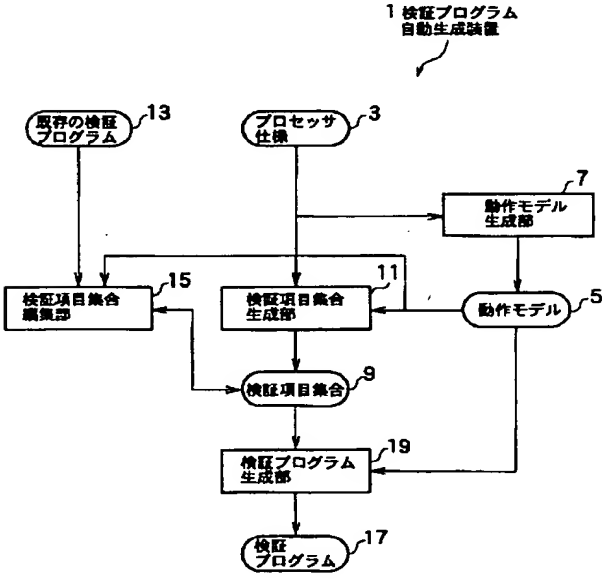
【図２７】各パイプライン・ステージに対するストール条件を表す表を示す図である。

【図２８】前段のパイプライン・ステージが存在しないパイプライン・ステージに対する次サイクルの命令を表す式を示す図である。

【図２９】前段および後段のパイプライン・ステージが

存在するパイプライン・ステージに対する次サイクルの命令を表す式を示す図である。
【図３０】後段のパイプライン・ステージが存在しないパイプライン・ステージに対する次サイクルの命令を表す式を示す図である。
【図３１】各パイプライン・ステージに対する次サイクルの命令を表す表を示す図である。
【図３２】図２の命令セット・アーキテクチャ仕様および図３のパイプライン動作仕様で規定される動作モデルを有限状態機械で表現する時に用いる状態を示す図である。
【図３３】本発明の第２の実施の形態に係るプロパティ自動生成装置の構成を示すブロック図である。
【図３４】、図３３の動作モデル生成部の動作モデル生成とプロパティ生成部のプロパティ生成の処理手順を示すフローチャートである。
【図３５】ハザード状態の例を示す図である。
【図３６】ハザード解消状態の例を示す図である。
【図３７】各パイプライン・ステージの命令とＲＴＬ記述中のレジスタとの対応関係の例を示す図である。
【図３８】図３５および図３６に示した例に対するプロパティを示す図である。
【図３９】本発明の第３の実施の形態に用いるプロセッサ仕様の例を示す図である。
【図４０】本発明の第３の実施の形態に用いる検証対象動作の例を示す図である。
【図４１】本発明の第３の実施の形態の動作を示すフローチャートである。
【図４２】動作モデルの例を示す図である。
【図４３】ＲＴＬ記述の例を示す図である。
【図４４】クライアント名とレジスタ名の対応関係の表を示す図である。
【図４５】生成されたプロパティを示す図である。
【符号の説明】
１　検証プログラム自動生成装置
３、２３　プロセッサ仕様
５　動作モデル
７、２７　動作モデル生成部
９　検証項目集合
１１　検証項目集合生成部
１３　既存の検証プログラム
１５　検証項目集合編集部
１７　検証プログラム
１９　検証プログラム生成部
２１　プロパティ自動生成装置
２５　仕様解析部
２９　プロセッサの回路記述
３１　回路記述読み込み部
３３　記述解析部
３５　プロパティ
３７　プロパティ生成部
３９　検証対象動作
４１　ＲＴＬ記述
４３　プロパティ

【図１】




【図２】

| 命令 | dest reg | src reg | 命令動作 |
|---|---|---|---|
| nop | – | – | – |
| add | rd | rs st | rd←rs+rt |
| sub | rd | rs rt | rd←rs−rt |
| fadd | rd | rs rt | rd←rs+rt |
| fsub | rd | rs rt | rd←rs−rt |
| ld | rd | base offset | rd←memory[basa+offset] |

【図３】

| 命令 | パイプライン動作 |
|---|---|
| nop | FET |
| int | FET→ALU→WB |
| fpu | FET→FPU→WB |
| ld | FET→ALU→MEM→WB |

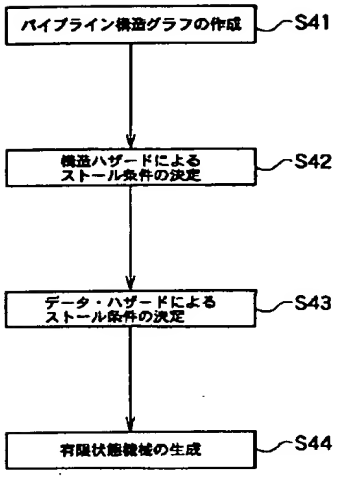
【図９】

Order(PSi) > Order(PSj) → PSi

【図６】

構造ハザード状態集合=[((ALU, int), (FPU, fpu))
((ALU, int), (MEM, ld))
((FPU, fpu), (MEM, ld))]

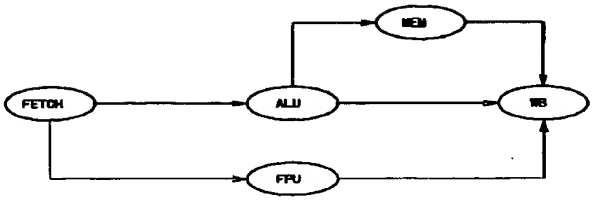
【図４】




【図７】

((ALU,int),(FPU,fpu)) → (ALU,int)
((ALU,int),(MEM,ld)) → (ALU,int)
((FPU,fpu),(MEM,ld)) → (FPU,fpu)

【図５】




【図８】

| パイプライン・ステージの状態 | 構造ハザード・ストール条件 |
|---|---|
| (ALU,int) | Inst(MEM) = ld \| Inst(FPU) = fpu |
| (MEM,ld) | 0 |
| (FPU,fpu) | Inst(MEM) = ld |

【図１５】

DHStallRule(Inst(FET),(PS,Inst(PS))) & (src(Inst(FET)) ∋ dest(Inst(PS)))

【図１０】

| パイプライン・ステージの状態 | 構造ハザード・ストール条件 |
|---|---|
| (ALU,int) | Order(Inst(ALU)) < Order(Inst(MEM)) \| Order(Inst(ALU)) < Order(Inst(FPU)) |
| (MEM,ld) | Order(Inst(MEM) < Order(Inst(ALU)) \| Order(Inst(MEM)) < Order(Inst(FPU)) |
| (FPU,fpu) | Order(Inst(FPU)) < Order(Inst(ALU)) \| Order(Inst(FPU)) < Order(Inst(MEM)) |

【図１１】

データ・ハザード集合 = {
(int,(ALU,int)),(int,(ALU,ld)),
(int,(FPU,fpu)),(Int,(MEM,ld)),
(int,(WB,int)),(int,(WB,fpu)),(int,(WB,ld)),
(fpu,(ALU,int)),(fpu,(ALU,ld)),
(fpu,(FPU,fpu)),(fpu,(MEM,ld)),
(fpu,(WB,int)),(fpu,(WB,fpu)),(fpu,(WB,ld)),
(ld,(ALU,int)),(ld,(ALU,ld)),
(ld,(FPU,fpu)),(ld,(MEM,ld)),
(ld,(WB,int)),(ld,(WB,fpu)),(ld,(WB,ld))
}

【図１８】

| PS | Inst | Next (PS, Inst) |
|---|---|---|
| FET | int | ALU |
| | fpu | FPU |
| | ld | ALU |
| ALU | int | WB |
| | ld | MEM |
| FPU | fpu | WB |
| MEM | ld | WB |

【図２４】

| PS | DHStall(PS) |
|---|---|
| WB | 0 |
| MEM | 0 |
| FPU | 0 |
| ALU | 0 |
| FET | 0 |

【図１２】

| | | ALU | | FPU | MEM | WB | | |
|---|---|---|---|---|---|---|---|---|
| | | int | ld | fpu | ld | int | fpu | ld |
| FET | nop | × | × | × | × | × | × | × |
| | int | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| | fpu | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| | ld | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

【図１３】

| | | ALU | | FPU | MEM | WB | | |
|---|---|---|---|---|---|---|---|---|
| | | int | ld | fpu | ld | int | fpu | ld |
| FET | nop | × | × | × | × | × | × | × |
| | int | ○ | ○ | ○ | ○ | × | × | × |
| | fpu | ○ | ○ | ○ | ○ | × | × | × |
| | ld | ○ | ○ | ○ | ○ | × | × | × |

【図１６】

Stall(PS) = SHStall(PS) | DHStall(PS) | ImplicitStall(PS)

【図１４】

| | | ALU | | FPU | MEM | WB | | |
|---|---|---|---|---|---|---|---|---|
| | | int | ld | fpu | ld | int | fpu | ld |
| FET | nop | × | × | × | × | × | × | × |
| | int | × | × | × | ○ | × | × | × |
| | fpu | × | × | × | ○ | × | × | × |
| | ld | × | × | × | ○ | × | × | × |

【図１７】

$$ImplicitStall(PS)=\begin{cases}0 & (n=0)\\ \sum_{NPS\in PSset}(Next(PS,Inst(PS))==NPS)\&Stall(Inst(NPS))) & (n\geq 1)\end{cases}$$

【図１９】

$$\begin{aligned}Next(PS,Inst)=\ &((PS==FET)\&(Inst(PS)==int)\&ALU)\\ &|((PS==FET)\&(Inst(PS)==fpu)\&FPU)\\ &|((PS==FET)\&(Inst(PS)==ld)\&ALU)\\ &|((PS==ALU)\&(Inst(PS)==int)\&WB)\\ &|((PS==ALU)\&(Inst(PS)==ld)\&MEM)\\ &|((PS==FPU)\&(Inst(PS)==fpu)\&WB)\\ &|((PS==MEM)\&(Inst(PS)==ld)\&WB)\end{aligned}$$

【図２０】

$$SHStall(PS)=\sum_{I\in InstSet}SHStall(PS,I)\&(Inst(PS)==I)$$

【図２１】

$$DHStall(PS)=\sum_{I\in InstSet}DHStall(PS,I)\&(Inst(PS)==I)$$

【図２２】

| PS | Inst | SHStall(PS,I) |
|---|---|---|
| WB | int | 0 |
| | fpu | 0 |
| | ld | 0 |
| MEM | ld | 0 |
| FPU | fpu | Inst(MEM)==ld |
| ALU | int | Inst(MEM)==ld \| Inst(FPU)==fpu |
| | ld | 0 |
| FET | int | 0 |
| | fpu | 0 |
| | ld | 0 |

【図２３】

| PS | SHStall(PS) |
|---|---|
| WB | 0 |
| MEM | 0 |
| FPU | Inst(MEM)==ld&Inst(FPU)==fpu |
| ALU | (Inst(FPU)==fpu) \| Inst(MEM)==ld) & Inst(ALU)==int |
| FET | 0 |

【図２５】

| PS | Inst | ImplicitStall(PS,Inst) |
|---|---|---|
| WB | int | 0 |
| | fpu | 0 |
| | ld | 0 |
| MEM | ld | 0 |
| FPU | fpu | 0 |
| ALU | int | 0 |
| | ld | 0 |
| FET | int | (Inst(MEM)==ld \| Inst(FPU)==fpu) & Inst(ALU)==int |
| | fpu | Inst(MEM)==ld & Inst(FPU)==fpu |
| | ld | (Inst(MEM)==ld \| Inst(FPU)==fpu) & Inst(ALU)==int |

【図２６】

| PS | ImplicitStall(PS) |
|---|---|
| WB | 0 |
| MEM | 0 |
| FPU | 0 |
| ALU | 0 |
| FET | (Inst(MEM)==ld \| Inst(FPU)==fpu) & Inst(ALU)==int<br>& Inst(FET)==int \| Inst(FET)==ld)<br>\| Inst(MEM)==ld & Inst(FPU)==fpu & Inst(FET)==fpu |

【図２７】

| PS | Stall(PS) |
|---|---|
| WB | 0 |
| MEM | 0 |
| FPU | Inst(MEM)==ld & Inst(FPU)==fpu |
| ALU | (Inst(MEM)==ld \| Inst(FPU)==fpu) & Inst(ALU)==int |
| FET | (Inst(MEM)==ld \| Inst(FPU)==fpu) & Inst(ALU)==int<br>& Inst(FET)==int \| Inst(FET)==ld)<br>\| Inst(MEM)==ld & Inst(FPU)==fpu & Inst(FET)==fpu |

【図２８】

$$NextInst(PS)=Stall(PS)\&Inst(PS)\ |\ \overline{Stall(PS)}\&Inst(IN)$$

【図２９】

$$NextInst(PS)=Stall(PS)\ \&\ Inst(PS)$$

$$\overline{Stall(PS)}\ \&\ \sum_{PPS\in PSSet}\overline{Stall(PPS)}\&(Next(PPS,Inst(PPS))==PS)\&Inst(PPS)$$

$$\overline{Stall(PS)}\ \&\ \prod_{PPS\in PSSet}Stall(PPS)\ |\ (Next(PPS,Inst(PPS))\neq PS)\&Bubble$$

【図３５】

(int , int , fpu , −, int)

【図３６】

(int , int , − , −, fpu)

【図３０】

$$\mathrm{NextInst(PS)} = \overline{\mathrm{Stall(PS)}} \,\&\, \sum_{\mathrm{PPS}\in\mathrm{PPSSet}} \overline{\mathrm{Stall(PPS)}}\&(\mathrm{Next(PPS,Inst(PPS))}==\mathrm{PS})\&\mathrm{Inst(PPS)}$$

$$\overline{\mathrm{Stall(PS)}} \,\&\, \prod_{\mathrm{PPS}\in\mathrm{PPSSet}} \mathrm{Stall(PPS)} \mid (\mathrm{Next(PPS,Inst(PPS))} \neq \mathrm{PS})\&\mathrm{Bubble}$$

【図３１】

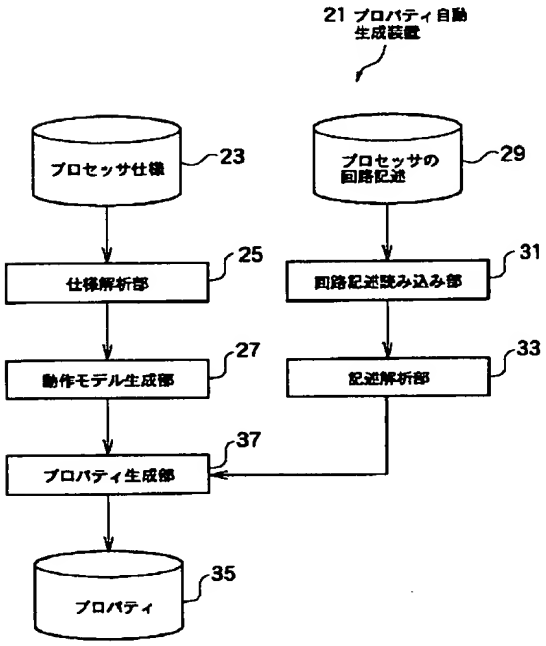
| PS | NextInst(PS) |
|---|---|
| FET | $\overline{\mathrm{Stall(FET)}}\&\mathrm{Inst(FET)} \mid \mathrm{Stall(FET)}\&\mathrm{Inst(IN)}$ |
| ALU | $\mathrm{Stall(ALU)}\&\mathrm{Inst(ALU)}$<br>$\mathrm{Stall(ALU)}\&\mathrm{Stall(FET)}\&(\mathrm{Inst(FET)}==\mathrm{INT} \mid \mathrm{Inst(FET)}==\mathrm{ALU})\&\mathrm{Inst(FET)}$<br>$\mathrm{Stall(ALU)}\&(\mathrm{Stall(FET)} \mid \overline{\mathrm{Inst(FET)}}==\mathrm{INT} \mid \mathrm{Inst(FET)}==\mathrm{ALU})\&\mathrm{Bubble}$ |
| FPU | $\mathrm{Stall(FPU)}\&\mathrm{Inst(FPU)}$<br>$\mathrm{Stall(FPU)}\&\mathrm{Stall(FET)}\&\mathrm{Inst(FET)}==\mathrm{FPU}\&\mathrm{Inst(FET)}$<br>$\mathrm{Stall(FPU)}\&(\mathrm{Stall(FET)} \mid \overline{\mathrm{Inst(FET)}}==\mathrm{FPU})\&\mathrm{Bubble}$ |
| MEM | $\mathrm{Stall(ALU)}\&\mathrm{Inst(ALU)}==\mathrm{LD}\&\mathrm{INST(ALU)}$<br>$\mathrm{Stall(ALU)} \mid \overline{\mathrm{Inst(ALU)}}==\mathrm{LD}\&\mathrm{Bubble}$ |
| WB | $\mathrm{Stall(FPU)}\&\mathrm{Inst(FPU)}==\mathrm{FPU}\&\mathrm{Inst(FPU)}$<br>$\mathrm{Inst(MEM)}==\mathrm{LD}\&\mathrm{Inst(MEM)}$<br>$!(\mathrm{Stall(ALU)}\mid \overline{\mathrm{Inst(ALU)}}==\mathrm{INT})$<br>$\&(\mathrm{Stall(FPU)}\mid \overline{\mathrm{Inst(FPU)}}==\mathrm{FPU})$<br>$\&\overline{\mathrm{Inst(MEM)}}==\mathrm{LD}\&\mathrm{Bubble}$ |

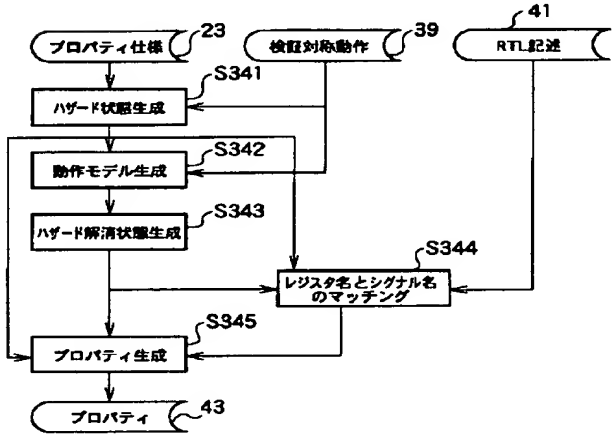
【図３２】

(Inst(FET),Inst(ALU),Inst(FPU),Inst(MEM),Inst(WB))

【図３３】

【図３４】




【図３９】

| アルゴリズム | クライアント |
|---|---|
| ラウンド・ロビン | CL0,CL1,CL2,CL3,CL4,CL5,CL6,CL7 |

【図３７】

【図３８】

| PS | Inst(PS) | RTL |
|---|---|---|
| FET | int | Fet_reg[3:0]="0001" |
| ALU | int | Alu_reg[3:0]="0001" |
| FPU | fpu | Fpu_reg[3:0]="0010" |
| WB | int | Wb_reg[3:0]="0001" |
| | fpu | Wb_reg[3:0]="0010" |

「Fet_reg[3:0]="0001",Alu_reg[3:0]="0001",Fpu_reg[3:0]="0010",Wb_reg[3:0]="0001"
にあるとき次のクロックサイクルで
Fet_reg[3:0]="0001",Alu_reg[3:0]="0001",Wb_reg[3:0]="0010"
になる」

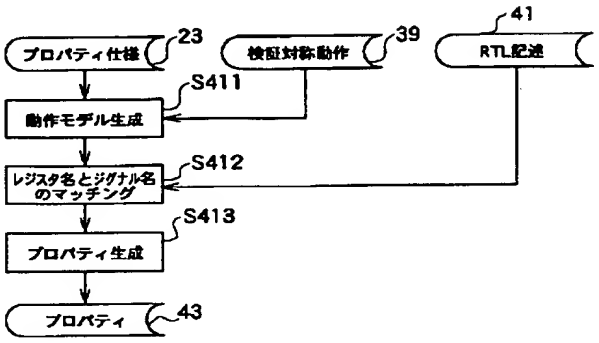
【図４０】

【図４２】

| クライアント数 | 8 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| クライアント | CL0 | CL1 | CL2 | CL3 | CL4 | CL5 | CL6 | CL7 |
| 優先順位 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

クライアントnが[illegible]、クライアントはバス要求を出した後、nクロック・サイクル内に許可される。

【図４１】




【図４３】

```
module round_robin(clk,request,grant)
input clk;
input [ 7:0 ] request;
output [ 7:0 ] grant;
reg [ 7:0 ] grant;
reg [ 7:0 ] slot;

  initial slot=0;

  always @ (posedge clk)
  begin
    grant=0;
    grant[slot]=request[slot];
    if (slot==7)
     slot=0;
    else
     slot=slot+1;
  end
endmodule
```

【図４４】

| クライアント名 | レジスタ名 | |
|---|---|---|
| | 使用要求 | 使用許可 |
| CL0 | request[0]=1 | grant[0]=1 |
| CL1 | request[1]=1 | grant[1]=1 |
| CL2 | request[2]=1 | grant[2]=1 |
| CL3 | request[3]=1 | grant[3]=1 |
| CL4 | request[4]=1 | grant[4]=1 |
| CL5 | request[5]=1 | grant[5]=1 |
| CL6 | request[6]=1 | grant[6]=1 |
| CL7 | request[7]=1 | grant[7]=1 |

【図４５】

「request[0]=1のとき、8クロック・サイクル内でgrant[0]=1」

---

フロントページの続き

(72) 発明者　水野　淳
神奈川県川崎市幸区堀川町580番１号　株式会社東芝半導体システム技術センター内

(72) 発明者　上谷　裕徳
神奈川県川崎市幸区堀川町580番１号　株式会社東芝半導体システム技術センター内

Ｆターム(参考)　2G032 AA03 AC08 AE12
5B013 CC01 EE08
5B046 AA08 BA03 JA05
9A001 DD03 HH32 LL08